Fully distributed and fault tolerant task management based on diffusions

Alain Bui[×] Olivier Flauzac[+]
Cyril Rabat[+]

[×] Laboratoire PRiSM (UMR CNRS 8144)
Université de Versailles-St-Quentin-en-Yvelines
45, avenue des Etats-Unis – F-78035 Versailles Cedex – France
alain.bui@prism.uvsq.fr

[+] Laboratoire CReSTIC
Université de Reims Champagne-Ardenne,
BP 1039, F-51687 Reims Cedex 2 – France
{olivier.flauzac,cyril.rabat}@univ-reims.fr

Abstract

The task management is a critical component for the computational grids. The aim is to assign tasks on nodes according to a global scheduling policy and a view of local resources of nodes. A peer-to-peer approach for the task management involves a better scalability for the grid and a higher fault tolerance. But some mechanisms have to be proposed to avoid the computation of replicated tasks that can reduce the efficiency and increase the load of nodes. In the same way, these mechanisms have to limit the number of exchanged messages to avoid the overload of the network.

In [4], we have proposed two methods for the task management called active and passive. These methods are based on a random walk: they are fully distributed and fault tolerant. Each node owns a local tasks states set updated thanks to a random walk and each node is in charge of the local assignment. Here, we propose three methods to improve the efficiency of the active method. These new methods

are based on a circulating word. The nodes local tasks states sets are updated thanks to periodical diffusions along trees built from the circulating word. Particularly, we show that these methods increase the efficiency of the active method: they produce less replicated tasks. These three methods are also fully distributed and fault tolerant. On the other way, the circulating word can be exploited for other applications like the resources management or the nodes synchronization.

1 Introduction

When a problem is submitted in a grid, it can be divided in a set of tasks. These tasks have to be assigned to nodes of the grid according to the resources needed for their computation and the resources owned and available on nodes. A grid has to manage a lot of resources as the storage space and the computational power, specific data, shared applications or tools. Each resource has to be identified for the global scheduling of tasks.

In *Nimrod* [6], the resources are gathered on an agent. So, the scheduling can be achieved with the agent knowledge and tasks are assigned by the agent. In Middleware *NetSolve* [3], the computational resources give an estimation of the task computation length in function of the task parameters. It lets the agent schedules the tasks according to the servers load. In such applications, the central agent is a critical point in the grid. An overload or a failure of this node can involve a grid failure or a low global efficiency. To improve the scalability, the global knowledge can be divided or shared in several servers or in a hierarchy of servers as in *Globus* [11] or in *DIET* [1].

Since several years, the peer-to-peer approach has been proposed for increasing the scalability and the fault tolerance of the applications. The centralization is avoided and the topology of the application is flexible: the deployment is also simplified. The authors of *CONFIIT* [9] choose to let the nodes in charge of the tasks selection. The tasks parameters are sent to all nodes so they can choose to compute a task according to their local resources. But the nodes of *CONFIIT* are setup in a virtual ring that has to be maintained. It involves a lot of control messages especially if the network is highly volatile and, in the worst case, the ring cannot be maintained.

To avoid the maintenance of a virtual structure, we proposed fully distributed methods based on a random walk. In [4] we propose a task man-

agement: the tasks parameters are sent to nodes and the tasks states are updated thanks to the circulation of a token. We proposed two tasks assignment methods called *passive* and *active*. For the passive method, the nodes wait for the token before selecting a task. The number of replicated tasks is low but some computational resources is unused. For the active method, the nodes select a task before receiving the token. It induces replicated tasks but it increases the global efficiency. In the following, we focus on the improvement of the active method. Indeed, we show in [4], that this method has a better efficiency than the passive method.

In this article, we propose to use another tool called the circulating word. Its aim is to collect node identities and to build and maintain spanning trees of the network. We propose to diffuse periodically the tasks states along these spanning trees in order to speed up the update of the nodes local knowledge.

In next section, we present the tools we used in our algorithm: the grid model, the random walks and the circulating word. In section 3, we present the task management (task definition, efficiency). Then, we present our solutions and we describe the algorithms. We show in section 5 some experimental results. Finally, we conclude and we present our future works.

2 Preliminaries

A model for grid In [14], we propose a model composed of 5 layers to analyze grid applications. The three lower layers concern the network, the routing and the messages exchange protocols. Layer 4 represents the resources management for the grid and the last one the other grid components (scheduling, monitoring, ...). The task management we expose here is for Layer 5. But a grid is built over four other layers and we have to take care of their impacts. We show that we can model a grid by a *directed* graph $G = (V, E)$, where V is a set of active nodes of the grid with $|V| = n$ and E is the set of directed communication links. An active node is a resource or a node that uses resources (to compute task). In the following, we use the terms "resource", "node" and "active node" interchangeably.

A communication link (i, j) exists if and only if j is a neighbor of i in the grid, i.e. i can directly send a message to j. Every node i can distinguish all its links of communication and maintains a set of neighbors denoted N_i. We consider that all resources of the grid have a distinct identity (IP address,

for example or a complete description with a specific language like *RSL* [7], in that case an indexation is needed to have better performances).

As G is a communication graph, we assume it is strongly connected. Indeed, if the graph is not strongly connected at a time, there exists a sink subgraph $E(G)$: resources of $G \backslash E(G)$ cannot be reached from any node of $E(G)$. For a token circulation, it means that the token will stay in $E(G)$ and cannot reach nodes of $G \backslash E(G)$. With our method, we accept that the graph stays not strongly connected during a short time. If this transient state is too long, unreachable resources will be considered as disconnected.

Random walk A random walk is a sequence of nodes visited by a token that starts at i and visits other resources according to the following transition rule: if the token is at i at time t then at time $t+1$, it will be at one of the neighbors of i, chosen uniformly at random among N_i ([13]). Similarly to deterministic distributed algorithms, the time complexity of random walk based token circulation algorithms can be viewed as the number of "steps" it takes for the algorithm to achieve the network traversal. With only one walk at a time (which is the case we deal), it is also equal to the message complexity. The cover time C — the average time to visit all nodes in the system — and the hitting time denoted by h_{ij} — the average time to reach a node j for the first time starting from a given node i — are two important values that appear in the analysis of random walk-based distributed algorithms. Both of them are on average bounded by n^3. There are three properties about random walks: *percussion* – an arbitrary node is visited in a finite time, *coverage* – all nodes are visited in a finite time and *meeting* – several random walks will meet each other in a finite time.

Circulating word A circulating word is a tool used to collect data on a network. It has been introduced in [12] for the detection of the execution termination of distributed algorithms. Particularly, it can be used to collect identities of visited nodes by a random walk. With a specific management, as proposed in [8], we are able to build a spanning tree rooted on the node that owns the token. This tree is perpetually updated and adapted to the topology when node failures occur. We can send data through the network along this tree. This application is used in our task management methods.

3 Task management

We define a task by the tuple $\{id_T, id_E, p, s, r\}$ where id_T is the task identity, id_E the identity of the task emitter, p the parameters of the task, s the state of the task and r the results of the task (if it has been computed). A task can be in three states: *uncomputed*, *in progress* (locally or in a distant node) and *computed*.

With a centralized method, the tasks are gathered on a server as we present on Figure 1 (a). When a task is assigned to a node, its state is modified and the task cannot be assigned to another node (excepted if a voluntary replication is achieved as in *BOINC* [2]). With our method, the parameters and results of the tasks are diffused to all nodes. On each node, a local set noted $\mathcal{E}$ contains a local view of the tasks states. When a node wants to compute a task, it selects an uncomputed task at random in its local set and tags it as *in progress*. This new state will be updated on other nodes step by step by the token (Figure 1 (b)). In the same way, when a node has to submit new tasks, it add them only to its local set. The tasks will be known by other nodes when the token will visit the node and will diffuse their parameters to other nodes.

When a task is selected, its state is *uncomputed* but another node can select simultaneously the same task. So, we obtain a *replicated task*. This involves a waste of computational power and decreases the efficiency of the task management. To compute the efficiency of the task management, we compare the sequential execution time t_e – that is the time for one node to compute all the tasks – and the distributed execution time t_d – that is the time for all nodes to compute all the tasks. Efficiency of the method, noted e, is obtained by the formula: $e = \frac{t_s}{t_e \times n} \times 100$, where n is the number of nodes. We also compare the task management solutions on the number of exchanged messages.

4 Diffusion based task management

We propose three solutions based on the active method described in [4]. A token circulates at random in the network and updates the local tasks states sets of nodes. We add a circulating word in the token that is used to build spanning trees. We describe three solutions to increase the active method efficiency: method $\mathcal{D}_s$ is based on periodical diffusions, method $\mathcal{D}_f$ is based

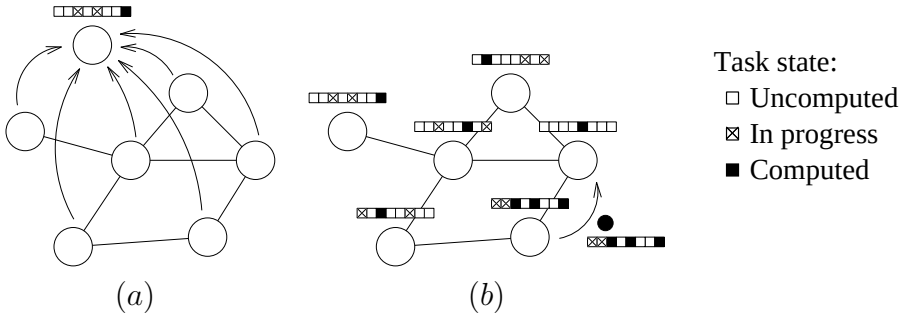


Figure 1: Centralized task management (a) and token based task management (b).

on diffusions with feedbacks and method $\mathcal{D}_m$ is based on diffusions with feedbacks followed by another diffusions.

4.1 $\mathcal{D}_s$ method

We define a token by a tuple $T = \{id_T, \mathcal{E}_T, W, C_T\}$ where id_T is the token identity, $\mathcal{E}_T$ is the tasks states set, W is a circulating word and C_T is a hop counter. When a node receives a token, it increments C_T and updates W (by adding its identity). If C_T is upper than a bound, noted b, the node builds from the circulating word a diffusion tree $\mathcal{T}_D$ rooted on it. Then, a diffusion of $\mathcal{E}_T$ (updated by the node set) is launched through $\mathcal{T}_D$. We define each message of the diffusion by the tuple $M_D = \{id_T, \mathcal{E}_M, \mathcal{T}_D\}$ where id_T is the token identity (used to control the message validity), $\mathcal{E}_M$ is a tasks states set updated according to visited nodes and $\mathcal{T}_D$ the diffusion tree. When a message is received, the node updates its local tasks states set and forwards the message to all of its neighbors in $\mathcal{T}_D$. To reduce the messages size, $\mathcal{T}_D$ can be reduced to the subtree rooted on neighbors.

The frequency of diffusion launches depends on bound b. To reduce useless diffusions, we compute b according to the current state of the global computation. Indeed, when the number of tasks to compute decreases, the local task selection method involves replicated tasks. So, we compute b ac-

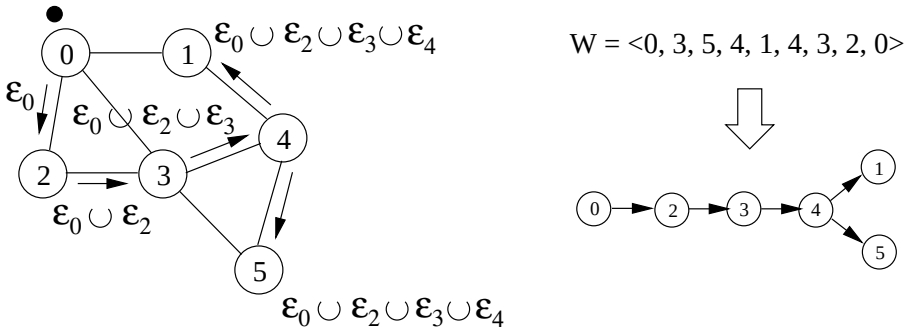


Figure 2: Example of a tasks states diffusion from a circulating word

cording to the following formula:

$$b = \min \left\{ \frac{nbT}{n} * c_r, m_r \right\}$$

nbT is the number of tasks to compute, n is the number of nodes, c_r is the refresh coefficient and m_r is the minimum refresh value. When the ratio between the number of tasks and the number of nodes becomes too small, the frequency of diffusions increases: more diffusions are launched to reduce replicated tasks. To prevent the overload of the network, we specify a minimum threshold noted m_r. c_r interacts on the diffusions frequency.

Example 1 *Figure 2 shows an example of a single diffusion. Node 0 receives the token and here, we suppose C_T becomes upper than b. Node 0 builds a diffusion tree from the circulating word and launches a diffusion. We show that local tasks states sets $\mathcal{E}$ are updated during the diffusion.*

4.2 $\mathcal{D}_f$ method

On the previous example, we observe that the nodes on leaves of the diffusion tree are updated with several tasks states sets but the set of the diffusion tree root (*i.e.* the initiator node) is not updated. In the same way, if a node is deeper in the tree, it is updated with several nodes sets but its local set is not diffused to other nodes. In Figure 2, Node 5 is updated with sets of Nodes 0, 2, 3 and 4 but its tasks states set is not sent to other nodes.

To improve the update of local sets, we propose to add a feedback after the diffusion. Each node sends its local set to its father. At the end of the diffusion and the feedback, the initiator node receives a global view of the tasks states. To limit the number of exchanged messages, we exploit the algorithm of the distributed recursive waves described in [10]. Before sending its set to its father, each node waits for the response of each son. To support node or link failures, we add a timeout on each node reseted at each diffusion: if a node does not receive the responses of its sons before the timeout ends, it sends its set to its father. If sons responses are received later, they will be ignored (or only used to update the node).

In the worst case, the diffusion and the feedback take $2 \times (n-1)$ steps. If b is lower than this value, several diffusions can be launched at the same time but on different diffusion trees. We need to identify each diffusion and nodes have to keep in memory their father in the diffusion and its sons. We add a diffusion counter in the token that is incremented at each diffusion. Each message of a diffusion is tagged by the counter value. On the nodes, we add two sets to keep in memory the node father and its sons corresponding to a diffusion. When a node has received a response of each son of a diffusion, it can send its own response to its father.

4.3 $\mathcal{D}_m$ method

After a diffusion with a feedback, the tasks states set of the initiator is updated. The nodes on the leaves of the tree are only updated by few nodes, especially if the diffusion tree has a small depth. So, after a diffusion with a feedback, we can initiate a new diffusion with the initiator set. After this diffusion, all the nodes will have the same view of the tasks states. The same diffusion tree can be used that does not cost any extra computational power for the initiator.

4.4 Example

Figure 3 shows an example of an update diffusion. Figure (a) presents the tasks states set of each node. Some tasks are considered as uncomputed on some nodes and are in progress on others nodes. A node receives the token and initiates a diffusion (Figure (b)). The tasks states sets are updated along the diffusion tree and the nodes on leaves of the tree obtain the best view.

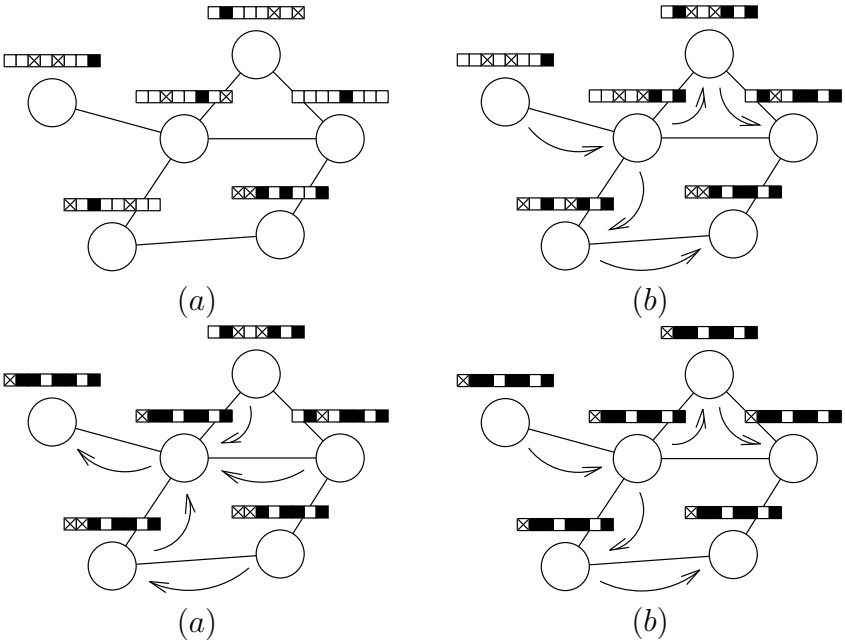


Figure 3: Diffusion based task management: task states are updated thanks to a diffusion through a tree built from a circulating word.

After the feedback (Figure (c)), the initiator receives the more recent view of all tasks. The last diffusion (Figure (d)) involves that all the nodes have the same view.

5 Experimental results

We simulate our methods with *Dasor* library [5]. We generate a set of random task lengths thanks to the log-normal law. We obtain a set of irregular task lengths. The set is sent to each node and we compute the time for the nodes to compute the tasks: here you suppose that all nodes have the same computational power. For the diffusion methods, we fix arbitrarily $c_r = 1000$ and $m_r = 1500$ (these values give a good compromise between the efficiency and the number of exchanged messages).

The first series of simulations presented on Figure 4 (a) shows the evolution of the efficiency in function of the grid nodes number. It presents the

executions results with 1000 nodes and a number of tasks that evolves from 1000 to 20000. We remark that the diffusion methods have a better efficiency than the active method especially when the tasks number increases (about 5% for $\mathcal{D}_m$ in average). For a small number of tasks, the efficiencies are almost identical. Indeed, at the beginning of the execution, each node selects at random a task in its local set and several nodes select the same tasks (the ratio is 1 task per node).

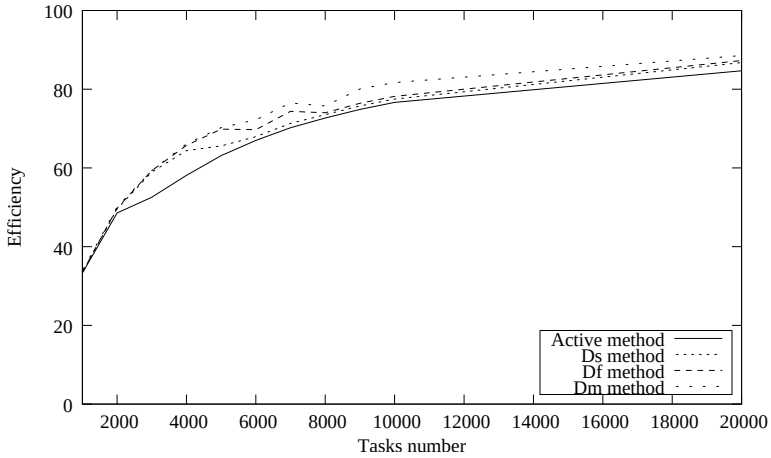
Figure 4 (b) presents executions results with a set of 20000 tasks and a number of nodes that evolves from 1000 to 5000. When the number of nodes increases, the diffusions methods have a better efficiency (more than 15% for $\mathcal{D}_m$). We observe that the efficiency decreases faster with the active method. The cover time of the token is higher and involves more replicated tasks: the latency between updates is higher. The diffusion methods seem to be more scalable.

On Figure 5 (a), we present the number of produced messages during the computation and on Figure 5 (b) the number of replicated tasks. We fix the number of nodes at 1000 and we increase the number of tasks from 1000 to 20000. We can observe that the number of messages for Method $\mathcal{D}_m$ is about twice more than the active method (with $c_r = 1000$ and $m_r = 1500$). About the number of replicated tasks, we have about twice less replicated tasks than the active method. The diffusion methods reduce the global load of the grid by reducing the useless computations.

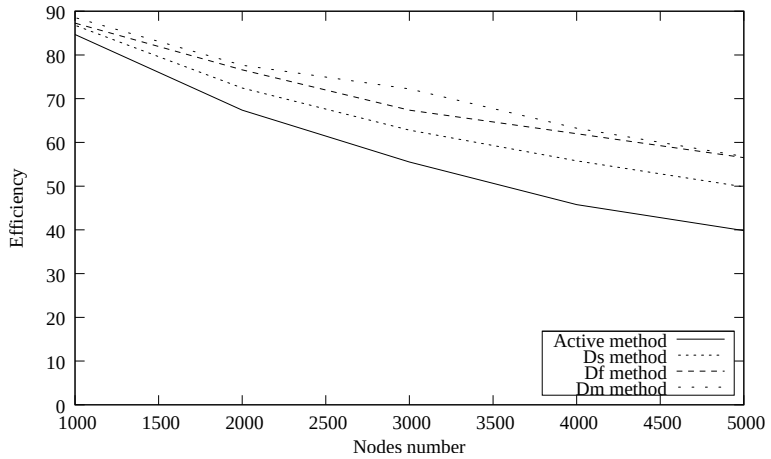
To improve the efficiency, c_r and m_r can be reduced to increase the frequency of the diffusions but it induces more messages. With these experimental results, we observe that we have already a better efficiency and less replicated tasks than for the active method. We also realize others simulations series with $c_r = 100$ and $m_r = 100$. For 1000 nodes and 20000 tasks, it increases the efficiency by 5% and reduce the number of replicated tasks. But it produces 4 times more messages than with the previous coefficients.

6 Conclusion

We propose in this article three solutions for the task management based on a random walk and a circulating word. The nodes are in charge of the local assignment of tasks according to their resources and the tasks states are updated thanks to the token. These solutions are fully distributed and

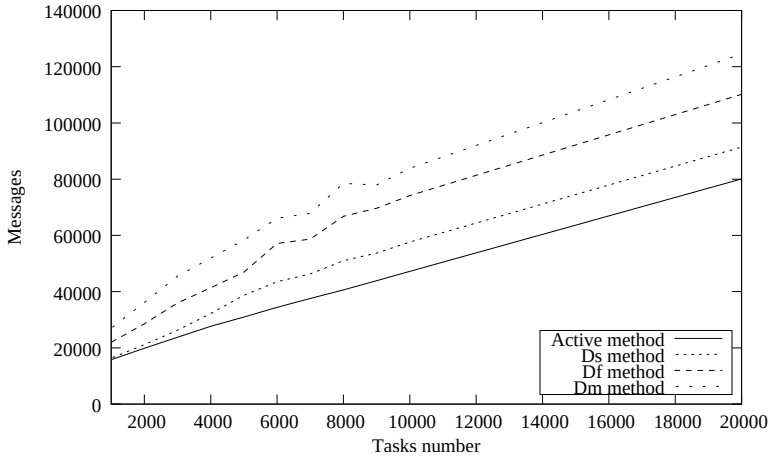


(a)

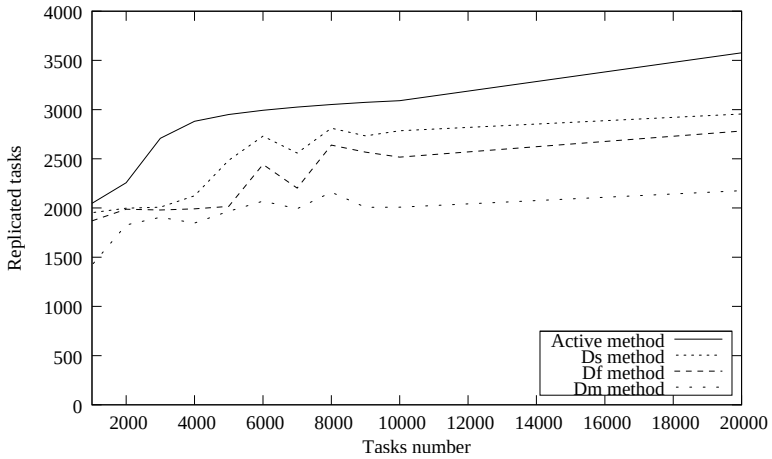


(b)

Figure 4: Evolution of the efficiency in function of the tasks number (a) and in function of the nodes number (b) .



(a)



(b)

Figure 5: Evolution of the messages exchanges (a) and of the replicated tasks (b) in function of the tasks number.

are resilient to node failures. We present some experimental results and we observe a better efficiency than the active method presented in [4]. These new methods produce more messages but they reduce significantly the number of replicated tasks: it reduces the useless load of grid nodes.

These methods are based on two coefficients c_r and m_r that allow to modify the frequency of diffusions according to the current state of the system (number of tasks and number of nodes). We plain to automatize these parameters to take into account others grid parameters as the bandwidth of the network or the actual load of the system. Another solution is to exploit the hybrid method proposed in [4] coupled with the diffusion methods. When the ratio between the number of nodes and the number of tasks is low, we may reduce the replicated tasks.

Acknowledgments

This work was partly supported by "Romeo"[1], the high performance computing center of the University of Reims Champagne-Ardenne.

References

[1] A. Amar, R. Bolze, A. Bouteiller, P. K. Chouhan, A. Chis, Y. Caniou, E. Caron, H. Dail, B. Depardon, F. Desprez, J.-S. Gay, G. Le Mahec, and A. Su. DIET: New Developments and Recent Results. In L. et al. (Eds.), editor, *CoreGRID Workshop on Grid Middleware (in conjunction with EuroPar2006)*, number 4375 in Lecture Notes in Computer Science, pages 150–170, Dresden, Germany, Aug. 2006. Springer Verlag.

[2] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[3] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[4] A. Bui, O. Flauzac, and C. Rabat. Fully Distributed Active and Passive Task Management for Grid Computing. In *ISPDC '07: Proceedings of the*

[1] `http://www.romeo2.fr/`

Sixth International Symposium on Parallel and Distributed Computing. IEEE Computer Society, July 2007.

[5] A. Bui, O. Flauzac, and C. Rabat. Dasor, a grid model based simulation library. In *I2CS'08, 8th International Conference on Innovative Internet Community Systems*. IEEE Computer Society, 2008. To appear.

[6] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. *CoRR*, cs.DC/0009021, 2000.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 62–82. Springer-Verlag, 1998.

[8] O. Flauzac. Random circulating word information management for tree construction and shortest path routing tables computation. In *On Principle Of DIstributed Systems*, pages 17–32. Studia Informatica Universalis, 2001.

[9] O. Flauzac, M. Krajecki, and J. Fugere. CONFIIT : a middleware for peer to peer computing. In *ICCSA 2003*, volume 2669 of *LNCS*, pages 69–78. Springer-Verlag, 2003.

[10] G. Florin, R. Gómez, and I. Lavallée. Recursive distributed programming schemes. In *International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 122–128, 1993.

[11] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *J. Comput. Sci. Technol.*, 21(4):513–520, 2006.

[12] I. Lavallée. *Algorithmique distribuée et parallèle*. Hermes, 1990.

[13] L. Lovasz. Random walks on graphs: A survey. In *Combinatorics: Paul Erdos is Eighty (vol. 2)*, pages 353–398. Janos Bolyai Mathematical Society, 1993.

[14] C. Rabat, A. Bui, and O. Flauzac. A random walk topology management solution for grid. In *II2CS*, volume 3908 of *LNCS*, pages 91–104. Springer-Verlag, 2006.